

Inhaltsverzeichnis

1	Testgetriebene Entwicklung	1
1.1	Was ist testgetriebene Entwicklung?	2
1.2	Woher kommt testgetriebene Entwicklung?	3
1.3	Wie funktioniert testgetriebene Entwicklung?	5
1.4	Warum sollte man testgetrieben entwickeln?	5
1.5	Voraussetzungen für testgetriebene Entwicklung	8
	1.5.1 Werkzeuge	8
	1.5.2 Methoden	9
1.6	Testgetriebene Entwicklung und Unit-Tests	11
1.7	Vor- und Nachteile	12
	1.7.1 Vorteile	12
	1.7.2 Nachteile	16
1.8	Anforderungen an einen Unit-Test	18
	1.8.1 Codequalität	19
	1.8.2 Unabhängigkeit	19
	1.8.3 Dokumentation	19
	1.8.4 Ressourcen	20
	1.8.5 Nur ein Testfall pro Test	20
1.9	Zusammenfassung	21
2	Frameworks	23
2.1	Die Frameworks im Überblick	23
2.2	Clientseitige Frameworks	24
2.3	QUnit	24
2.4	Jasmine	29
2.5	Nachteile clientseitiger Frameworks	33
2.6	Serverseitige Frameworks	34
2.7	JsTestDriver	34
2.8	Karma	41
2.9	Zusammenfassung	44

3	Workshop: Red, Green, Refactor	45
3.1	Die Aufgabenstellung	45
3.2	Konzeptarbeit	46
3.3	Setup	47
3.3.1	Dateistruktur	48
3.4	Die ersten Schritte	49
3.4.1	Red – der erste Test	49
3.4.2	Green – der Test läuft erfolgreich ab	50
3.5	Der nächste Schritt	51
3.5.1	Red – mehr Einsicht	51
3.5.2	Green – fake it	52
3.5.3	Refactor – dynamischer Rückgabewert	53
3.6	innerSpace – ein Teilproblem	53
3.6.1	Red – ein erster Test für innerSpace	53
3.6.2	Green – Implementierung der innerSpace-Methode	54
3.6.3	Refactor – Duplikate reduzieren	54
3.7	Erweiterung der innerSpace-Methode	55
3.7.1	Red – Triangulation	55
3.7.2	Green – erweiterte Fake-it-Lösung	56
3.7.3	Refactor – innerSpace für alle Buchstaben	57
3.8	Erklärende Tests	58
3.8.1	Grenzfälle testen	58
3.9	Fehlerfälle abtesten	59
3.9.1	Red – innerSpace soll eine Exception werfen	59
3.9.2	Green – Exception werfen	60
3.9.3	Refactor – den gültigen Wertebereich definieren	61
3.10	outerSpace	62
3.10.1	Red – Leerzeichen in outerSpace	62
3.10.2	Green – fake it – outerSpace	63
3.10.3	Red – Triangulation von outerSpace	63
3.10.4	Green – Erweiterung der outerSpace-Methode	64
3.10.5	Refactor – dynamische Version von outerSpace	64
3.11	Auslagerung von Funktionalität	65
3.11.1	Red – die getIndexOf-Methode	66
3.11.2	Green – Implementierung der getIndexOf-Methode	66
3.11.3	Refactor – dynamische Version der getIndexOf-Methode	67
3.11.4	Red – Fehlerbehandlung innerhalb der getIndexOf-Methode	67
3.11.5	Green – Integration der Fehlerbehandlungsroutine	68
3.11.6	Refactor – Integration der getIndexOf-Methode	68

3.12	Fehlerbehandlung in der outerSpace-Methode	70
3.12.1	Red – Test für die Fehlerbehandlung in outerSpace	70
3.12.2	Green – erfolgreiche Fehlerbehandlung in der outerSpace-Methode	71
3.12.3	Refactor – Anpassung der Fehlerbehandlung in outerSpace	71
3.13	Eine Zeile des Diamanten	72
3.13.1	Red – ein Test für eine Zeile	72
3.13.2	Green – Ausgabe einer statischen Zeile	73
3.13.3	Red – ein zweiter Test für eine Zeile	73
3.13.4	Green – dynamische Ausgabe einer Zeile	74
3.13.5	Red – die erste und letzte Zeile	74
3.13.6	Green – die erste und letzte Zeile	75
3.13.7	Refactor – die erste und letzte Zeile	75
3.14	Zusammenführung der Komponenten	76
3.14.1	Red – Test für die obere Hälfte des Diamanten	76
3.14.2	Green – upperHalf gibt den korrekten Wert zurück	77
3.14.3	Refactor – Umbau der upperHalf-Methode	77
3.14.4	Red – Test für die lowerHalf-Methode	78
3.14.5	Green – die lowerHalf-Methode gibt einen statischen Wert zurück	79
3.14.6	Refactor – Erweiterung der lowerHalf-Methode	79
3.15	Der letzte Schritt – die Integration	80
3.15.1	Red – Test für einen vollständigen Diamanten	80
3.15.2	Green – fake it der toString-Methode	81
3.15.3	Refactor – finale Implementierung der toString-Methode	81
3.16	Refactorings	82
3.16.1	Refactoring #1 – charCodeAt	82
3.16.2	Refactoring #2 – upperHalf und lowerHalf	82
3.17	Zusammenfassung	85
4	Testinfrastruktur	87
4.1	Funktionsweise	87
4.1.1	Die Serverkomponente	88
4.1.2	Manuelle Testausführung	89
4.1.3	Der Browser	90
4.2	Workflow	92
4.3	Debugging innerhalb der Testumgebung	94
4.4	System mit Fehlertoleranz	96
4.5	Zusammenfassung	97

5	Spies, Stubs und Mocks	99
5.1	Sinon.JS	99
5.1.1	Installation und Konfiguration	100
5.1.2	Test der Installation	100
5.2	Jasmine	101
5.3	Test Doubles	101
5.4	Spies	103
5.4.1	Wann kommen Spies zum Einsatz?	103
5.4.2	Spies verwenden	104
5.4.3	Die Spy-Schnittstelle	106
5.4.4	Spies im konkreten Beispiel	107
5.4.5	Spies in Jasmine	111
5.5	Stubs	112
5.5.1	Wann kommen Stubs zum Einsatz?	112
5.5.2	Stubs verwenden	114
5.5.3	Die Stub-Schnittstelle	114
5.5.4	Stubs im konkreten Beispiel	115
5.5.5	Stubs in Jasmine	117
5.6	Mocks	118
5.6.1	Wann kommen Mocks zum Einsatz?	118
5.6.2	Mocks verwenden	119
5.6.3	Die Mock-Schnittstelle	119
5.7	Zusammenfassung	120
6	Abhängigkeiten vom DOM	121
6.1	Abhängigkeiten	121
6.2	Fixtures	122
6.3	Selbst erstellte HTML Fixtures	124
6.3.1	Die Aufgabenstellung	124
6.3.2	Setup	125
6.3.3	Ein einfacher Test	125
6.3.4	HTML Fixture	126
6.3.5	Green – Anzeige der Nachrichten	127
6.3.6	Triangulate	128
6.3.7	Cleanup	129
6.3.8	Green – dynamischer Validator	130

6.4	jasmine-jquery	131
6.4.1	Installation	131
6.4.2	Fixtures laden	133
6.4.3	Zusätzliche Matcher	137
6.5	Karma html2js	137
6.6	Zusammenfassung	139
7	Asynchrones Testen und Kommunikation mit dem Server	141
7.1	Asynchrone Funktionen	141
7.1.1	Ein erstes asynchrones Beispiel	142
7.1.2	Asynchronität mit Promises	143
7.1.3	Promises mit Q	144
7.1.4	Promises testen	145
7.2	Zeitabhängige Funktionen	149
7.2.1	Problemstellungen bei zeitabhängiger Programmierung	149
7.2.2	Einsatz von Fake-Timern	151
7.2.3	Abhängigkeit vom Datum	153
7.3	Abhängigkeiten vom Server	154
7.3.1	Problemstellung bei der Kommunikation mit dem Server	155
7.3.2	Tests mit Abhängigkeit vom Server	156
7.3.3	Einsatz von Fake-Servern	157
7.4	Zusammenfassung	159
8	Tests in neuen und in bestehenden Applikationen	161
8.1	Neue Applikationen	162
8.2	Auswahl der Technologien	162
8.3	Setup der Umgebung	164
8.3.1	Konzeption und Anforderungen	165
8.3.2	Erste Tests	166
8.3.3	Weiteres Vorgehen nach dem ersten Test	167
8.4	Bestandscode	168
8.4.1	Testgetriebene Entwicklung im Bestandscode	169
8.4.2	Problemstellungen im Bestandscode	169
8.4.3	Umgebung in bestehenden Applikationen	170
8.4.4	Strategien für die Erstellung von Tests	170
8.4.5	Testgetriebene Entwicklung neuer Features	171
8.4.6	Testgetriebene Entwicklung bei der Überarbeitung von Quellcode	174
8.5	Zusammenfassung	177

9	Testen von Node.js-Applikationen	179
9.1	Serverseitige Entwicklung mit Node.js	179
9.1.1	Installation	180
9.1.2	Betrieb	181
9.1.3	Der NPM	182
9.2	Testframeworks für Node.js	182
9.2.1	Assert	183
9.2.2	Nodeunit	184
9.2.3	Mocha	185
9.2.4	Weitere Testframeworks für Node.js	186
9.3	Testgetriebene Entwicklung mit Node.js	186
9.3.1	Konzeption	187
9.3.2	Installation von expect.js	187
9.3.3	Struktur und erster Test	188
9.3.4	Umsetzung der Businesslogik	190
9.3.5	Integration	191
9.4	Test Doubles in Node.js	193
9.4.1	Sinon.js	194
9.4.2	nock	195
9.4.3	mockery	196
9.5	Zusammenfassung	198
10	Tools, die das Testen einfacher machen	199
10.1	Die Entwicklungsumgebung	199
10.1.1	WebStorm	200
10.2	Code Coverage	202
10.2.1	Installation des Coverage-Plug-ins	203
10.2.2	Konfiguration des Coverage-Plug-ins	203
10.2.3	Der Coverage-Report	204
10.2.4	WebStorm und Code Coverage	206
10.2.5	Ignorieren von Quellcode	207
10.2.6	Nachteile der Code Coverage	208
10.3	Grunt und Gulp	209
10.3.1	Installation von Grunt	210
10.3.2	Testen mit Grunt	211
10.3.3	Installation von Gulp	212
10.3.4	Testen mit Gulp	213
10.4	Zusammenfassung	215
	Stichwortverzeichnis	217